Starting your own AI agents is a powerful way to automate tasks, interact with users, or perform reasoning and decision-making.

Here's a step-by-step guide tailored to your level of technical skill and goals.

□ Step 1: Define Your Goal

Before coding, decide:

- What is the agent's purpose? (e.g., customer support, data scraping, coding assistant)
- What kind of tasks will it do? (chat, plan, search, automate apps)

Examples:

- A personal AI assistant that schedules meetings.
- A research agent that reads PDFs and summarizes them.
- A stock market monitor that analyzes trends.

□ Step 2: Choose Your Foundation (Framework)

Here are some powerful open-source frameworks to build AI agents:

1. LangChain (Python)

- Great for building LLM-powered workflows.
- Ideal for document analysis, multi-agent conversations, tools, etc.
- Website: https://www.langchain.com

2. AutoGen by Microsoft

- Multi-agent collaboration framework.
- Ideal for task-oriented LLM agent teams.
- GitHub: https://github.com/microsoft/autogen

3. CrewAI

- Newer multi-agent framework focused on task decomposition.
- Easy to define agents with roles and goals.
- GitHub: https://github.com/joaomdmoura/crewAI

4. AutoGPT / AgentGPT

- Autonomous, goal-driven agents.
- Can browse the web, write code, etc.
- More experimental and less deterministic.

□ Step 3: Choose a Language Model (LLM)

Options:

- **OpenAI GPT-4 / GPT-4o** (most powerful for reasoning + tools)
- **Anthropic Claude** (great for long documents)
- **Mistral / LLaMA** (open-source, fine-tuneable)
- **Google Gemini** (if you're in that ecosystem)

Free/open options:

- **Ollama**: Run models like LLaMA3, Mistral locally.
- **LM Studio**: GUI for local LLMs.

□ Step 4: Set Up Tools

AI agents often need tools like:

- **Search engine API** (Bing, Google Custom Search)
- **Code interpreter** (Python REPL)
- **Calendar or email API** (Google Calendar, Gmail)
- **Browser automation** (Selenium, Puppeteer)

Frameworks like LangChain and AutoGen handle this modularly.

□ Step 5: Build a Simple Agent

Here's a basic LangChain agent example in Python:

```
from langchain.agents import initialize_agent, load_tools
from langchain.llms import OpenAI

llm = OpenAI(model="gpt-4", temperature=0)
```

```
tools = load_tools(["serpapi", "llm-math"], llm=llm)

agent = initialize_agent(tools, llm, agent="zero-shot-react-description",
verbose=True)

agent.run("What's the weather in Tokyo and how many days until Christmas?")
```

□ Step 6: Iterate & Test

Start simple:

- Test your agent with logs and verbose outputs.
- Handle edge cases and failures.
- Think about memory — should it remember context or past actions?

□ Bonus: Add Memory & Autonomy

- **Memory**: Use vector stores (e.g., FAISS, Chroma) for long-term memory.
- **Planning**: Add planner + executor agents (LangChain, AutoGen support this).
- **Autonomy**: Let it evaluate and revise its plan based on results.

□ Step 7: Deploy Your Agent

- **Command-line**: Good for quick use.
- **Web app**: Use FastAPI, Flask, or Streamlit.
- **Chatbot**: Integrate with Telegram, Discord, or Slack.
- **Desktop/mobile**: Use Electron or React Native for wrappers.

To run that same code using **LLaMA 3** with **Ollama** instead of OpenAI GPT-4, you'll use the `Ollama` integration from `langchain_community.llms`. Below is the updated code using **LangChain + Ollama + LLaMA3**:

✅ Updated Code Using Ollama + LLaMA3

```
from langchain_community.llms import Ollama
from langchain.agents import initialize_agent, load_tools

# Load LLaMA 3 via Ollama
llm = Ollama(model="llama3", temperature=0)

# Load tools – make sure serpapi key is set as environment variable
SERPAPI_API_KEY
tools = load_tools(["serpapi", "llm-math"], llm=llm)

# Initialize agent
agent = initialize_agent(
    tools,
    llm,
    agent="zero-shot-react-description",
    verbose=True
)

# Run the agent
agent.run("What's the weather in Tokyo and how many days until Christmas?")
```

✅ Requirements

1. **Install Ollama + Models:**

   ```
   # Install Ollama if not already
   curl -fsSL https://ollama.com/install.sh | sh

   # Pull the LLaMA 3 model
   ollama pull llama3
   ```

2. **Install Python Packages:**

   ```
   pip install langchain langchain-community
   ```

3. **Set SERPAPI API Key (for Web Search Tool):**

   ```
   export SERPAPI_API_KEY=your_key_here
   ```

Great! Let's modify the code to use only **free, local tools** — so you **don't need a paid API** like SerpAPI or OpenAI.

We'll use:

- **Ollama with LLaMA3** (local LLM)
- **LLM Math Tool** (built-in LangChain tool for math reasoning)

Here's the updated code:

□ LangChain + Ollama + LLaMA3 with Free Tools

```
from langchain_community.llms import Ollama
from langchain.agents import initialize_agent, load_tools

# Load LLaMA3 model locally via Ollama
llm = Ollama(model="llama3", temperature=0)

# Only use free, local tools
tools = load_tools(["llm-math"], llm=llm)

# Initialize the agent
agent = initialize_agent(
    tools=tools,
    llm=llm,
    agent="zero-shot-react-description",
    verbose=True
)

# Run a simple task (no external web APIs)
response = agent.run("How many days until Christmas from today?")
print(response)
```

□ Setup Steps

1. Install Ollama & LLaMA3

```
curl -fsSL https://ollama.com/install.sh | sh
ollama pull llama3
```

2. Install Python dependencies

```
pip install langchain langchain-community
```

To convert your LangChain-based agent to **CrewAI**, we'll take a different approach — CrewAI is based on the concept of **"Agents" + "Tasks" + "Crew"**. Instead of `initialize_agent(...)`, we explicitly define each role and assign it a task.

□ Same Logic Using CrewAI + Ollama (LLaMA3) + Free Tools

First, install dependencies:

```
pip install crewai langchain langchain-community
```

Then, here's your **CrewAI version** of the same logic:

```
from langchain_community.llms import Ollama
from crewai import Agent, Task, Crew

# Set up the local LLaMA3 model via Ollama
llm = Ollama(model="llama3", temperature=0)

# Define a simple agent who can do math or reasoning
math_agent = Agent(
    role="Date and Math Analyst",
    goal="Answer questions about dates and perform basic calculations",
    backstory="You're a helpful assistant great at reasoning and solving
time-based problems.",
    verbose=True,
    llm=llm
)

# Define the task for the agent
task = Task(
    description="Figure out how many days are left until Christmas from
today.",
    agent=math_agent
)

# Set up the crew with one agent and one task
crew = Crew(
    agents=[math_agent],
    tasks=[task],
    verbose=True
)

# Kick off the execution
result = crew.kickoff()
print(result)
```

□ What's Happening Here:

- We're manually defining an **Agent** (with a role and goal).
- Assigning it a **Task**.
- Putting them into a **Crew** to run.
- **No paid APIs** required — just your local `llama3` model.

□ Output

It should return something like:

"There are 156 days until Christmas."

(LLaMA3 will do its best using the current date it "thinks" it is; you can add exact logic with tools if you want precision.)

Great! Let's expand your CrewAI setup to include two new agents:

- □ **Planner Agent** – Plans how to approach the problem.
- □ **Researcher Agent** – Assists with logic, date reasoning, and background.
- □ **Analyst Agent** – Calculates how many days until Christmas.

□ Full CrewAI Example with Planner, Researcher, and Analyst (all using LLaMA3 via Ollama)

```
from langchain_community.llms import Ollama
from crewai import Agent, Task, Crew

# Load LLaMA3 locally via Ollama
llm = Ollama(model="llama3", temperature=0)

# --- Define Agents ---

# 1. Planner Agent
planner = Agent(
    role="Planner",
    goal="Break down complex goals into clear, actionable steps",
    backstory="You're an expert at thinking through problems and organizing
tasks for others.",
    verbose=True,
    llm=llm
)
```

```
# 2. Researcher Agent
researcher = Agent(
    role="Researcher",
    goal="Support the team by providing relevant knowledge and reasoning
about dates and calendars",
    backstory="You specialize in digging up important info and providing
logical insights, especially around timelines and holidays.",
    verbose=True,
    llm=llm
)

# 3. Analyst Agent
analyst = Agent(
    role="Date and Math Analyst",
    goal="Calculate how many days are left until Christmas from today",
    backstory="You handle all number-related tasks and date calculations.",
    verbose=True,
    llm=llm
)

# --- Define Tasks ---

# Task 1: Planning
plan_task = Task(
    description="Create a clear plan to determine how many days are left
until Christmas from today.",
    agent=planner
)

# Task 2: Research
research_task = Task(
    description="Based on the planner's outline, provide details about
today's date and when Christmas occurs. Include any relevant insights.",
    agent=researcher,
    depends_on=[plan_task]  # Waits for planner
)

# Task 3: Final Calculation
analysis_task = Task(
    description="Using the planner's plan and the researcher's findings,
calculate exactly how many days are left until Christmas.",
    agent=analyst,
    depends_on=[plan_task, research_task]  # Waits for both
)

# --- Create the Crew ---
crew = Crew(
    agents=[planner, researcher, analyst],
    tasks=[plan_task, research_task, analysis_task],
    verbose=True
)

# --- Run the Crew ---
result = crew.kickoff()
print("\n Final Answer:", result)
```

□ Setup Reminder

Make sure you've done:

```
pip install crewai langchain langchain-community
ollama run llama3
```

□ Output You Can Expect

Your planner will outline steps, the researcher will explain how to get today's date and when Christmas is, and the analyst will do the math to figure out the days left. LLaMA3 handles the logic — all **fully local** and **free**.